Ex0 - Server and Client

You are to make two programs, a client and a server. The server will be for multiple clients and start and wait for connections. The client will take an IP address as a command line argument and attempt to connect to the server.

Once connected a client will begin periodically sending an integer value that it is incrementing. The server will take all received integers and add it to its own integer variable. Once the server’s integer reaches a certain value, the server will end itself. This functionality will be provided to you through clientMain.c and serverMain.c. Your assignment is to create the underlying networking functions. You will turn in 2 files, server.c and client.c.

The networking will be using ipv4 addresses. The sockets should all be non-blocking.

Your assignment will be graded using copies of the clientMain.c and serverMain.c provided to you. Feel free to edit and change yours however you want, but I would suggest you keep a fresh copy to run your own testing on.

You will be responsible for making server.c and client.c according to the following specifications. You must include all structs and functions detailed below. However you are free to make any extra structs or functions that you believe will help you complete the assignment.

**Allowed Headers**

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <stdbool.h>

#include <sys/types.h>

#include <fcntl.h>

**Client.c**

**typedef struct Client {**

**int sock;**

**struct timeval tv;**

**} Client;**

**#define BUFF 1024**

macro defining buffer size.

**extern Client \*c;**

**Client \*connectAsClient(char \*ip);**

Here you will initialize the Client struct. **sock** will be a file descriptor of a socket that was made in the domain **AF\_INET** and with the type of **SOCK\_STREAM**. **timeval** can be set to **0** seconds. We will want to set ip a **sockaddr\_in** to hold out ip address information. We want this **sockaddr\_in** to connect to the provided ip address. It will be of type **AF\_INET**. Then we want to connect our socket with the ip address.

**int receiveData(Client \*c, void \*buffer);**

checks if there is data to be read from the socket. If there is, it reads it into the buffer and returns the size of the data read from the client’s socket.

It returns -1 if there is any issue.

**Server**

**typedef struct Server {**

**int sock;**

**int maxClients;**

**int \*clientSocks;**

**int addrlen;**

**struct sockaddr\_in addr;**

**struct timeval tv;**

**} Server;**

**#define BUFF 1024**

macro defining buffer size.

**extern void \*welcomeMessage;**

**extern int welcomeSize;**

**extern Server \*s;**

**Server\* setUpConnections();**

This function will initialize the server struct. **clientSocks** will be initialized to the size of **maxClients** (at least 10). **timeval** can be set to **0** seconds. **sock** will be initialized with the file descriptor you get from creating a socket. We will be using the communication domain **AF\_INET.** The socket will be of type **SOCK\_STREAM**.

You will probably want to set socket options to allow multiple connections. **addr** will be set to use any address.

The function will bind the socket to a local port and listen to it. Any errors must be detected and the process must be aborted by returning a null pointer. Success will return a pointer to the server struct.

**void closeServer(Server \*s);**

Frees all allocated memory of the Server struct and closes all of the sockets being used.

**void serverSendReceive(Server \*s, void \*buffer, int \*gotData);**

This is the largest bulk of the Server’s work. Here it will receive incoming client connections and add them to **clientSocks**. When a client is added, the server will send them the **welcomeMessage**, which can be initialized to 0 or whatever value you want.

The server will also check for incoming data from its clients and send this to any other clients that are connected. If a client disconnects it should be removed from **clientSocks**.

Here it will be useful to use **FD\_SETS** and **select()** to keep track of connections. Remember we want this to be non-blocking!

**Compiling and Testing**

You will be provided a Makefile for the code. Feel free to use it. This is what the grader will test with. Your code will also be tested with **valgrind** for any memory leaks.

You can set up your server first and test it using telnet.